Projet Logiciel Transversal

Battle city

LI Yijie

Talbe des matières

1 Objectif

1.1 *Présentation générale*

Le jeu Battle City consiste en une bataille de tanks. Le terrain est composé d'obstacles(comme un labyrinthe). Chaque tank est muni d'un canon et doit tirer et toucher les tank adverses.

1.2 *Règles du jeu*

Le jeur se joue par un seul joueur.

Pour diriger le tank, il faut utiliser les flèches directionnelles Haut Bas Gauche Droite pour se diriger le tank de joueur, et le boutton F pour tirer. Le joueur doit tuer toutes les tanks des ennemis. Les tanks adverses jouent automatiquement(IA) et ils essaient de détruire la base de joueur(un oiseau).
Le jeu finit si la base est détruit ou le joueur est touée.

2 Description et conception des états

2.1 *Description des états*

Le jeu est formé par un ensemble d'éléments fixes (mur en bois et mur en fer) et un ensemble d'éléments mobiles (tanks). Tous les éléments les coordonnées et identifiant de type d'élément.

Le class **StaticElement**: il comporte des murs en différentes types (wood,iron,sea)

Le class **MobileElement**: il représente les unités de tank. Ces unités partagent beaucoup d'attributs et de fonctions. Elles ont toutes de la vie, une vitesse de déplacement.

A l'ensemble des éléments statiques et mobile, on rajoute quelque propriétés:

Epoque: il représente temps correspondant à l'état du jeu.

Vitesse: le nombre d'époque par seconde.

Computer d'unités: le nombre d'unités d'une même équipe

2.2 *Conception logiciel*

Dans le diagramme des classes pour les états (Figure 1), on a quelque classes:

Class Element: Toute les classes filles d'Element représenter les catégories d'élément.

La fabrication d'éléments: On utilise la classe ElementFactory pour fabriquer d'éléments.

Conteneurs d'éléments: On met toutes les éléments dans ElementList et ElementGrid. ElementList a une list des toutes les éléments et ElementGrid peut gérer une grille.

2.3 *Conception logiciel: extension pour le rendu*

Observateurs de changements. Dans le diagramme de classes, on présente les classes permettant a des tiers de réagir lorsqu'un événement se produit dans l'un des éléments d'état.
Les observateurs implantent l'interface StateObserver pour être avertis des changements de propriétés d'état. Pour connaître la nature du changement, ils analysent l'instance de StateEvent. La conception de ces outils suit le patron Observer.

2.4 *Conception logiciel: extension pour le moteur de jeu*

Dans quelque classes par rapport à Element, on ajoute des méthodes de clone() et equals().

Méthode clone(): elle peut faire la copie de l'instance
Méthode euqals(): elle peut faire la comparaison entre deux instances.

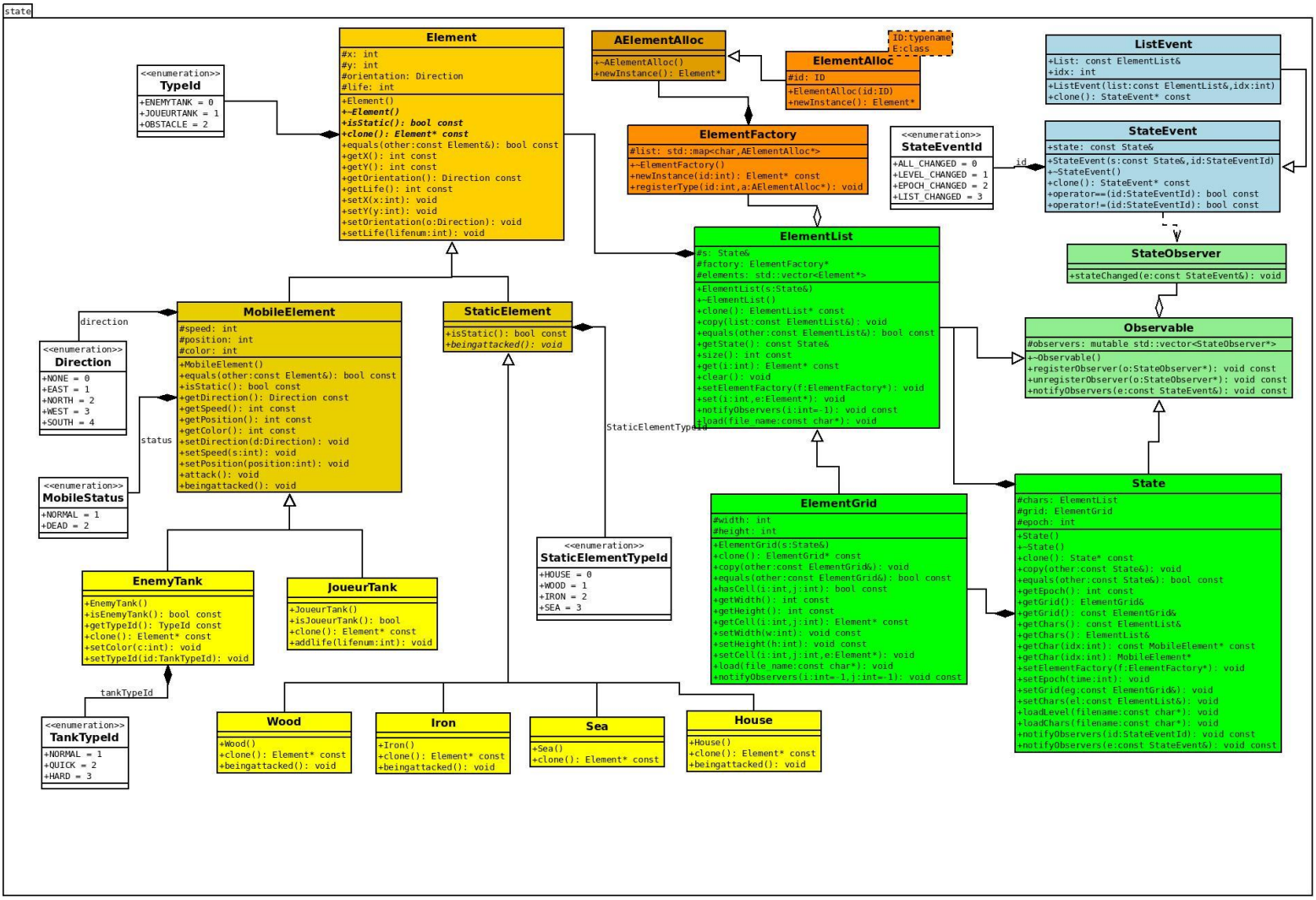

Figure1: Diagramme de classes de l'état

3. Rendu : Stratégie et Conception

3.1 *Stratégie de rendu d'un état*

Pour le rendu d'un état, nous avons opté pour une stratégie assez bas niveau, et relativement proche du fonctionnement des unités graphiques. En effet, les cartes graphiques actuelles, tout comme celles des années 80, sont plus efficace si le CPU prépare les éléments à rendre au sein de structures élémentaires, avant de tout envoyer au GPU.

Plus précisément, nous découpons la scène à rendre en plans (ou « layers ») : un plan pour le niveau (mur, pastilles, etc.), un plan pour les éléments mobiles (pacman, fantômes) et un plan pour les informations (vies, scores, etc.). Chaque plan contiendra deux informations bas-niveau qui seront transmises à la carte graphique : une unique texture contenant les tuiles (ou « tiles »), et une unique matrice avec la position des éléments et les coordonnées dans la texture. En conséquence, chaque plan ne pourra rendre que les éléments dont les tuiles sont présentes dans la texture associée.

3.2 *Conception logiciel*

Le diagramme des classes pour le rendu est en Figure 4.

On utilise la bibliothèque SFML, on a choisi d'implémenter l'interface sf::drawable à render::scene et ses layers. Cette implémentation permet de simplifier les appels graphiques.

Layer

Un layer(plan) est un objet réalisant le lien entre une liste d'éléments et leur position spatiale dans la fenêtre ainsi que leur représentation(texture). Layer hérite de sf::drawable permet de dessiner out le layer.

Scene

Elle contient tous les plants, et son implémentation de draw() appelle les draw() de ses layers.

Elle possède une fonction update(ElementList), qui est appelée lors d'une modification de l'état(pattern Observer)

Tile

Cet objet est simplement une représentation de la clef et les informations de texture associée.

3.3 *Conception logiciel: extension pour les animations*

Animations. Les animations sont gérées par les instances de la classe Animation. Chaque plan tient une liste de ces animations, et le client graphique fait appel aux méthodes de mise à jour pour faire évoluer ses surfaces.
Pour le cas des animations de mouvement, par exemple lorsqu'un tank se déplace, on a ajouté toutes les informaitons permettant d'afficher le déplacement sans dépendre de l'état.

3.4 *Ressources*

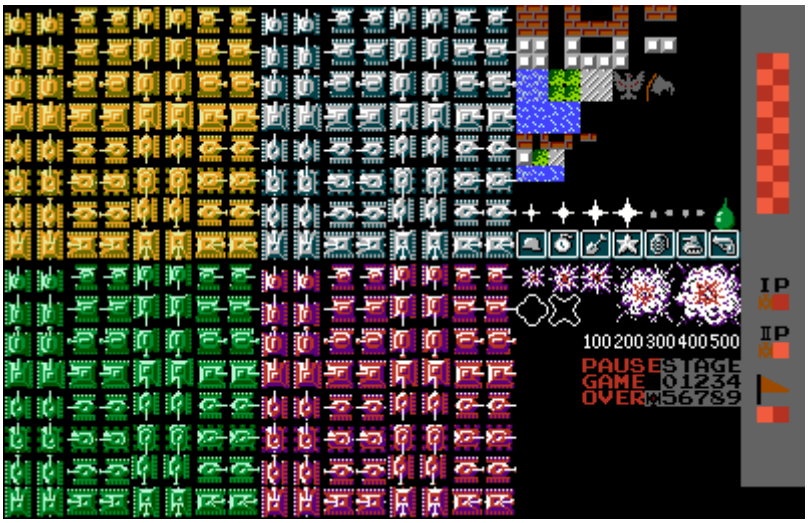

Figure 2: Exemple de texture

3.5 Exemple de rendu

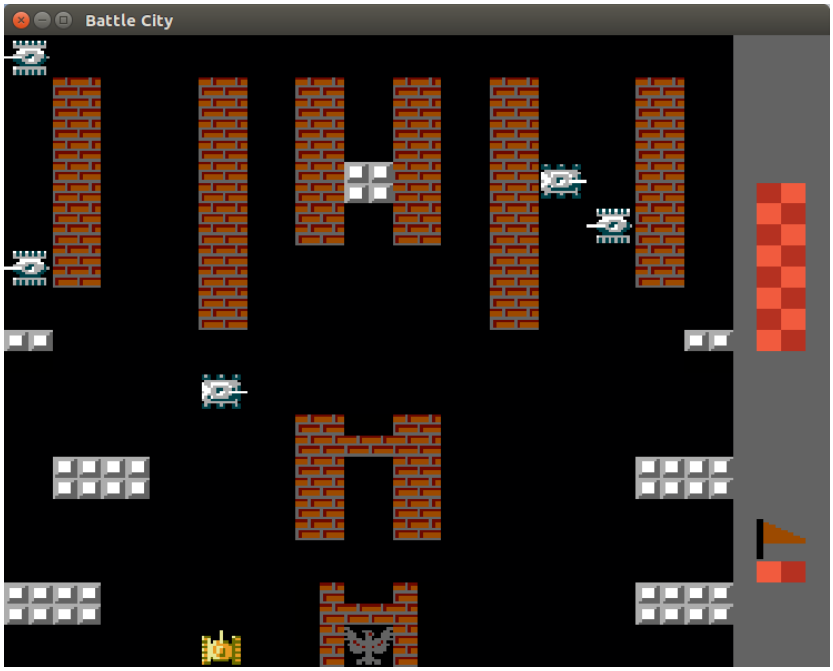

Figure 3 : Exemple de rendu

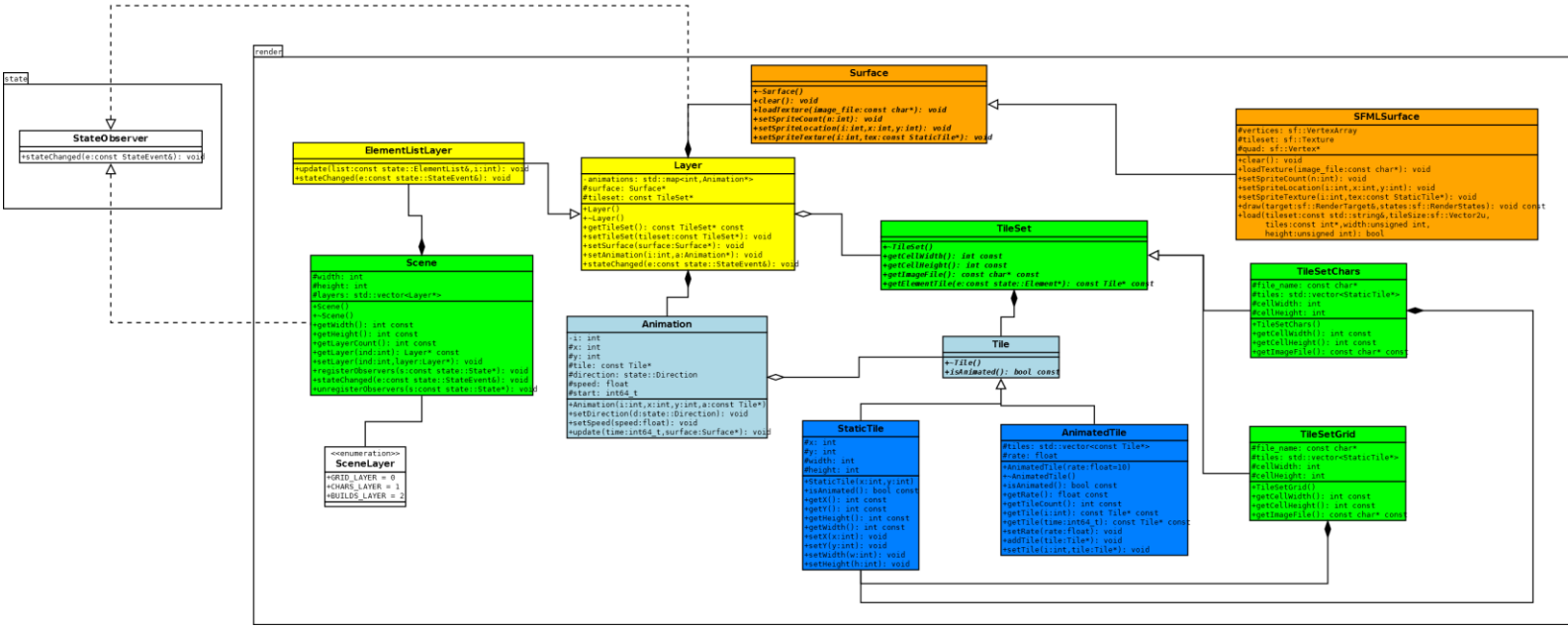

Figure 4 : Diagramme de classes du moteur de rendu